

# Análise Comparativa de Sistemas de Controle de Versões Baseados em Código Aberto

Erivan de Sena Ramos<sup>1</sup>, Rejane Cunha Freitas<sup>2</sup>

<sup>1</sup> Faculdade Integrada do Ceará (FIC), Fortaleza, CE, Brasil, erivansr@gmail.com

<sup>2</sup> Faculdade Integrada do Ceará (FIC), Fortaleza, CE, Brasil, rejanecf@fic.br.

## Resumo

Este artigo apresenta a pesquisa realizada para avaliar os sistemas de controle de versões de código aberto: CVS e *Subversion*; no intuito de analisar suas características utilizadas em projetos de desenvolvimento de *softwares*. Inicialmente foi construído um referencial teórico abordando os conceitos de controle de versões, destacando a importância do uso de sistemas de controle de versões pela gestão de configuração. Também foram expostas as principais características presentes em um sistema de controle de versões, dando enfoque aos sistemas analisados. Em seguida foram realizados testes, que resultaram em um comparativo, o qual adotou critérios estabelecidos pelo RUP para escolha de ferramentas de suporte ao desenvolvimento de *softwares*. **Palavras Chaves** - Controle de Versões. Código Aberto. Gestão de Configuração. Projetos de *Software*.

## Abstract

This article presents a research to evaluate systems of version control of open source: CVS and *Subversion*, and intends to analyze the characteristics used in the software project development. To start with, a theoretic basis was drawn up dealing with concepts of version control pointing out the importance of using systems of version control by configuration management. It was also highlighted the main characteristics present in a system of version control focusing the systems analyzed. Tests done afterwards resulted in a comparison which adopted requirements established by RUP, for the choice of supporting tools in the development of softwares.

**Keywords** - Version Control. Open Source. Configuration Management. Software Projects.

## 1. Introdução

O controle de versões de *software*, uma das principais atividades da gestão de configurações<sup>1</sup>, surgiu para controlar a evolução dos sistemas de *software* [11], possibilitando principalmente a recuperação de versões anteriores, o desenvolvimento paralelo e a auditoria do desenvolvimento [22].

Os esforços feitos no intuito de desenvolver as primeiras atividades neste campo têm registro por volta de 1950, época em que os controles de versões eram realizados de forma manual, onde o código-fonte de programas era armazenado no formato de cartões

perfurados, sendo assim possível recuperar as versões de acordo com a necessidade [2].

Na década de 70, foi lançado o sistema *Source Code Control System (SCCS)*, utilizado para controlar o desenvolvimento de código-fonte [3], sendo considerado como a primeira ferramenta de controle de versões [9]. No início da década de 80, foi desenvolvido o *Revision Control System (RCS)*, um sistema utilizado para gerenciar revisões de documentos texto, em particular programas fonte, documentação e dados de teste [3]. O RCS foi o primeiro sistema a implementar muitos dos princípios de controle de versões utilizados até hoje pelas mais modernas ferramentas [21]. A partir dos anos 90, novos sistemas começaram a serem desenvolvidos, sendo melhoradas as técnicas de armazenamento, controle de acesso e integração com outras ferramentas [10].

Atualmente, existem diversos sistemas de controle de versões, alguns inclusive com código aberto<sup>2</sup>, um dos mais conhecidos é o *Concurrent Versions System (CVS)*, cujo desenvolvimento vem sendo feito desde o final da década de 80. O CVS é baseado no RCS, entretanto, ao contrário deste, gerencia também a estrutura de diretórios [22], foi desenvolvido por Dick Grune em 1986, e consistia originalmente em um conjunto de scripts *shell* do UNIX. Em 1989, ele foi projetado e codificado na linguagem de programação C por Brian Berlinger, e posteriormente, Jeff Polk desenvolveu o suporte à *branches* [23]. A partir de 1995 foi idealizado o *Subversion*, com o intuito de ser um sistema de controle de versões *open source*, que pudesse suprir as limitações dos outros sistemas existentes na época [7]. Entretanto, só em 2001 o sistema foi desenvolvido por Karl G. Fogel e Ben Collins-Sussman [22] tendo sua versão 1.0 disponibilizada para o mercado apenas em 2004 [5].

Os sistemas *Subversion* e CVS, hoje em dia, são mantidos pelos projetos *Trigris.org* e *GNU*<sup>3</sup>, respectivamente. Ambos os projetos são constituídos por comunidades de usuários voluntários que desenvolvem projetos de *software* livre. Na visão de [13], apesar de haver uma diversidade de sistemas de controle de versões, essa área ainda é carente de pesquisa que faça uso de sua infraestrutura com o objetivo de atingir maior qualidade e produtividade no desenvolvimento de *software*. Mas recentemente, no intuito de investigar esse assunto, alguns pesquisadores [15]; [22]; [14]; [10]; [8]; vêm desenvolvendo importantes trabalhos que visam especificar, testar e

<sup>1</sup> A gestão de configuração é um dos enfoques da ITIL® - *Information Technology Infrastructure Library*. [4]

<sup>2</sup> Neste trabalho, será utilizado o termo “código aberto” como tradução do termo original em inglês *open source*. Eventualmente, o termo “*software* livre” também será utilizado.

<sup>3</sup> <http://subversion.tigris.org> e <http://savannah.nongnu.org/projects/cvs>.

desenvolver ferramentas que subsidiam o funcionamento dos sistemas de controle de versões em projetos de *software*, bem como automatizar operações, hoje realizadas pelo usuário.

Sabendo da importância dos sistemas de controle de versões para projetos de *softwares*, este artigo teve como objetivo analisar as principais características e funções apresentadas pelos sistemas de controle de versões de código aberto *CVS* e *Subversion*, as quais podem influenciar na escolha e adoção desses sistemas em projetos de softwares. Assim, espera-se que este estudo auxilie aos gerentes de configuração, na escolha de um sistema de controle de versões para uso no processo de desenvolvimento de *softwares*. Este artigo encontra-se dividido nas seguintes partes: um breve referencial teórico de aspectos importantes sobre sistemas de controle de versões, com uma abordagem voltada para o *CVS* e o *Subversion*, seguida da descrição da metodologia utilizada, apresentação do comparativo realizado entre os mesmos e a análise dos resultados.

## 2. Controle de versões em projetos de software

No entendimento de [24] o processo de mudança é uma realidade que ocorre durante todo o ciclo de vida de um *software*. Como as organizações solicitam as mudanças de forma muito aleatória, todas as mudanças devem ser registradas e efetuadas no sistema a um custo razoável, daí porque a necessidade de serem gerenciadas. Para [16], durante o processo de desenvolvimento de *softwares*, mudanças não planejadas podem levar rapidamente ao caos.

O controle de versões também é essencial nas atividades definidas no *CMMI - Capability Maturity Model Integration*, onde são estabelecidas metas à gerência de configuração para realizar o rastreamento e controle das mudanças dos itens de configuração, além de garantir a integridade dos mesmos, sendo necessário para isto, manter: *baselines*, histórico de revisões e mudanças [1]. Para [19] a gestão integrada do conhecimento e dos eventos ocorridos no projeto somado a um ambiente que permite o trabalho cooperativo e distribuído, proporcionado pelo controle de versões, pode tornar os processos de desenvolvimento de *softwares* mais eficientes.

O armazenamento de versões dos artefatos no desenvolvimento de *softwares* deve ser feito com a utilização de sistemas de controle de versões. Estes possibilitam ao gerente de configurações um melhor domínio sobre as operações realizadas [24], tais como: rastrear automaticamente os artefatos, garantir a integridade e registros históricos das operações durante o desenvolvimento do artefato (quem, quando, o quê e por que foi modificado) [8]. Portanto, em qualquer projeto de desenvolvimento de *software* moderno, um sistema de controle de versão é elemento essencial [15][17].

### 2.1. Mecanismos de sistemas controle de versões

A finalidade principal de um sistema de controle de versões é possibilitar aos usuários compartilharem os dados, permitindo uma edição colaborativa do *software* em desenvolvimento. Para isto é necessária a existência de um local que armazene todas as versões dos arquivos, denominado repositório.

No repositório são armazenadas as árvores de versões, estrutura lógica que mapeia todas as versões para determinado arquivo ou conjunto de arquivos. A analogia é feita com uma árvore porque existe um “tronco” e “ramos” criados a cada edição de uma versão [10]. O repositório é acessado pelos usuários por meio da *workspace*, espaço temporário para manter uma cópia local da versão a ser modificada, que isola as alterações feitas por um desenvolvedor das alterações paralelas, tornando a versão em desenvolvimento privada [12].

O desenvolvimento paralelo ou *branch*, encontrado na maioria dos sistemas de controle de versão, permite que várias pessoas trabalhem simultaneamente no mesmo objeto visionado, através de derivações feitas a partir de uma determinada versão. Segundo [8], esse tipo de evolução é utilizado para incorporar características particulares em determinadas versões ou também gerar versões de testes com modificações que poderão, ou não, ser integradas às demais versões.

A junção entre as versões realiza a fusão automática das versões, gerando uma única versão que agrega todas as alterações. Conforme [10], para este caso, a solução adotada por muitos sistemas de controle de versões é a realização de uma operação denominada *merge*, onde algoritmos calculam a diferença de linhas e criam uma versão final, integrando as alterações realizadas por diferentes desenvolvedores.

Além disso, outro mecanismo, neste caso, usado para a integração entre o repositório e a *workspace*, são as operações de sincronização. Algumas são definidas por [8]:

- *Check-out*: permite que uma determinada versão de artefato seja extraída do repositório para a *workspace*.
- *Check-in*: utilizada para criar o artefato no repositório pela primeira vez ou para criar uma nova versão do artefato quando este passar por uma modificação.
- *Lock*: bloqueia o artefato não permitindo que qualquer operação seja feita sobre ele até que o mesmo usuário realize o desbloqueio.
- *UnLock*: permite que o artefato seja desbloqueado pelo usuário que efetuou seu bloqueio.

Existem dois modelos de gerenciamento de repositório utilizados pelos sistemas de controle de versões: distribuído e centralizado. No modelo distribuído, cada desenvolvedor trabalha diretamente em uma cópia local de arquivo e a atualização da cópia no repositório compartilhado é feita ao final de todas as modificações desejadas. No modelo distribuído, um usuário possui um repositório próprio acoplado a uma *workspace*; para obter

versões dos outros usuários só é possível ao realizar as operações de sincronização entre os repositórios locais.

No modelo centralizado, também definido como cliente-servidor, as mudanças são feitas para um servidor central único. O modelo centralizado, aplicado no CVS e *Subversion*, em que os usuários trabalham de forma concorrente no repositório, é o mais utilizado atualmente [10]. Para controlar o acesso simultâneo aos nós de uma árvore de versões, existem dois métodos utilizados pelos sistemas de controle de versões: *lock-modify-unlock* (bloqueia-modifica-desbloqueia também conhecido por *exclusive lock*) e *copy-modify-merge* (copia-modifica-mescla também conhecido por *optimistic merge*) [20]. No método *lock-modify-unlock*, o repositório permite que apenas uma pessoa de cada vez altere o arquivo. Essa política de exclusividade é gerenciada usando *locks* (travas) [7].

Segundo [5], o método *lock-modify-unlock* apresenta problemas que podem tornar seu uso desaconselhado, pois o travamento de artefatos pode causar problemas administrativos, serialização de trabalho, problemas de *deadlock*, além da espera pelo desbloqueio causar perda de tempo. No método *copy-modify-merge*, os usuários trabalham simultaneamente e independentemente, modificando suas cópias privadas do artefato, que foram copiadas do repositório; depois as cópias privadas são fundidas numa nova versão [7].

O *Subversion*, o CVS, e outros sistemas de controle de versões usam um método de *copy-modify-merge* como uma alternativa ao *locking*. [7]. Ele funciona bem na prática e elimina os principais problemas apresentados pelo método *lock-modify-unlock* para o compartilhamento de arquivos entre os usuários [5]

Cabe destacar ainda a infraestrutura necessária para o controle de versões em um projeto de *software*, que segundo [5], é a presença de uma rede de computadores sobre o protocolo TCP/IP entre o servidor (estação central onde está situado o repositório) e os clientes (estações onde trabalham os desenvolvedores e onde estão localizadas as áreas de trabalho). No servidor, o sistema operacional pode ser qualquer um compatível com o sistema de controle de versões. Nas estações cliente, é necessário estar instalada a ferramenta cliente de controle de versões, que facilitará a utilização do sistema. Em meio as várias existentes no mercado pode-se citar o *Tortoise*<sup>4</sup>, um aplicativo *Windows*® que funciona sobre o *Windows Explorer*, e que segundo [5], tem interface e instrumentos internos que facilitam as operações básicas do fluxo de trabalho dos desenvolvedores, promove um menor tempo de aprendizado e uma maior produtividade.

No estudo de [5], cujo objetivo foi definir um plano de gerência de configuração, onde o controle de versões fosse um modelo de trabalho consistente, seguro e eficaz em projetos de desenvolvimento de *softwares* em equipe; mostrou que a implantação de uma política de controle de versão é factível e viável, podendo inclusive ser

implementada através de ferramentas de código aberto. Entretanto, demanda um nível de gestão proporcional ao tamanho dos projetos e da organização, requer treinamento de pessoal e algum esforço de adequação a novos processos.

### 3. Metodologia

Esta pesquisa é classificada como descritiva que na definição de [18] se destina a realizar uma avaliação dos dados que compõe o objeto de pesquisa. Tem como objetivo gerar conhecimentos que permitam caracterizar as alternativas para sistemas de controle de versões de código aberto em projetos de *software*, avaliando aspectos importantes por meio de uma análise comparativa, fundamentada em uma pesquisa bibliográfica e experimental. Do ponto de vista da abordagem, a pesquisa é qualitativa, pois buscou coletar informações sobre os aspectos dos sistemas comparados, para então apresentar a análise, produto da pesquisa. Visando atingir o objetivo desta pesquisa foram executados os seguintes passos:

- **Estudo:** pesquisa bibliográfica sobre aspectos de controle de versões e dos sistemas e ferramentas utilizadas na análise comparativa.
- **Instalação:** realizado *download* dos *softwares* a partir dos sítios oficiais e instalar as ferramentas utilizadas na pesquisa (CVS, *Subversion*, *TortoiseCVS* e *TortoiseSVN*).
- **Testes:** realizado os testes com arquivos genéricos, utilizando as ferramentas clientes para realizar as operações de sincronização de acordo com os critérios estabelecidos.
- **Análise:** os aspectos estudados nos sistemas de controle de versões foram apresentados em quadros comparativos, e realizada a análise.

### 4. Comparação entre os sistemas de controle de versões CVS e Subversion

A utilização dos sistemas de controle de versões CVS e *Subversion* na análise comparativa realizada neste trabalho se justifica por ambos serem *softwares* livres [13] de boa qualidade [20] e apresentarem popularidade no uso em projetos de *software* [10]; além de possuírem características essenciais para um melhor controle de versões, tais como: utilização de um repositório centralizado e aplicação do método de *copy-modify-merge*, os quais facilitam ainda mais o processo de controle de versões, pois as informações se tornam mais prontamente disponíveis aos usuários à medida que cada um vai modificando os arquivos e realizando o *commit* [23]; [5]; [7].

Na perspectiva de [20], devido à crescente necessidade do uso de sistemas de controle de versões e os custos restritivos dos sistemas comerciais, o CVS e o *Subversion*, se apresentam como soluções livres atraentes para as organizações que necessitam aperfeiçoar o processo de desenvolvimento. Embora ambos possuam características semelhantes, diferem quando se trata do estilo em que é realizado o histórico de versões no repositório. No CVS,

<sup>4</sup> <http://www.tortoisecvs.org> e <http://www.tortoisesvn.org>.

ao ser realizado um *commit* de um conjunto de arquivos, o controle de versões é feito por cada arquivo individualmente. Assim, o resgate de uma revisão, ou seja, o resgate de uma determinada versão de qualquer data mais antiga, num determinado momento, não fica alinhada em função da versão, onde cada arquivo produz uma nova versão no repositório e é armazenado como "uma fotografia" do momento. Já no *Subversion* o controle de versões é feito pelo conjunto de arquivos enviados ao servidor e, portanto há sempre uma versão global para todos os documentos. Toda revisão, em qualquer momento será sempre uma coluna alinhada.

#### 4.1. Definição dos critérios de comparação

Tabela 1  
Descrições dos critérios utilizados na análise

Crítérios	Descrição
Características e funções	A funcionalidade oferecida pelo sistema. Essa deve ser a conclusão geral da tabela 'Características e Funções'.
Integração	O nível de integração com outras ferramentas.
Aplicabilidade	Até que ponto oferece suporte ao processo de desenvolvimento
Extensibilidade	A capacidade de personalizar e configurar.
Suporte a equipe	A capacidade de oferecer suporte a uma equipe de usuários que está geograficamente distribuída.
Usabilidade	A facilidade de aprender e usar.
Maturidade	O nível de maturidade da ferramenta. Algumas organizações não comprariam uma versão 1, independentemente da boa reputação do sistema.
Suporte a configurações	Suporte disponível para instalação, configuração e uso.
Treinamento	Treinamento disponibilizado pelo fornecedor.

Os critérios adotados para a análise comparativa entre os sistemas *CVS* e *Subversion* apresentada neste trabalho são descritos na tabela 1; eles foram baseados em uma das atividades indicadas pelo *RUP* (*Rational Unified Process*)<sup>5</sup> para a fase de iniciação do projeto, a qual prevê selecionar e adquirir ferramentas de suporte necessário ao projeto de *software*. Segundo o *RUP*, para a uma melhor análise, além dos testes das características e funções apresentadas pelo produto, é necessário realizar a comparação de outros critérios necessariamente importantes para a utilização do sistema ou ferramenta no projeto de *software*.

Para o critério características e funções foram considerados alguns itens, expostos na tabela 2, definidos e utilizados por alguns pesquisadores [6]; [14]; em análises comparativas de sistemas empregados pela gerência de configuração em projetos de *softwares*.

Tabela 2  
Descrições das características e funções comparadas

Características/ Funções	Descrição
<i>Commit</i> atômico	Garantir a atomicidade das atualizações no

<sup>5</sup> A descrição das atividades e processos do *RUP*® são disponibilizados em <http://www.wthree.com/rup/portugues/index.htm>.

	repositório.
Controle de diferentes tipos de arquivos	Além do ASCII, permitir controle de arquivos em diversos formatos tais como Unicode e binário.
Controle de mudanças	Prover informações sobre quais itens estão em alteração por algum membro da equipe.
Controle de permissões	Atribuir de permissões individuais de leitura e gravação aos membros de equipe.
<i>Merges e Braches</i>	Permitir realizar junções e realizar de maneira satisfatória os <i>braches</i> (Quando a linha de desenvolvimento precisa ser dividida em duas ou mais).
Portabilidade	Suportar a integração de itens oriundos de clientes com ambientes operacionais distintos.
Renomeações de arquivos	Permitir renomear arquivos no repositório.
Suporte	Identificar distintas versões de arquivos que compõem uma distribuição e recuperação da mesma.

#### 5. Apresentação e análise dos dados

Para os testes unitários foram instalados e configurados os sistemas de controle de versões em um servidor com *Linux Ubuntu* como sistema operacional, e na estação cliente, a qual utiliza *WindowsXP* como sistema operacional, foram instalados as ferramentas clientes *TortoiseSVN* e *TortoiseCVS* para o uso do *Subversion* e *CVS*, respectivamente. Após as citadas instalações e configurações dos mesmos, foram realizados testes com arquivos genéricos contendo código-fonte, realizando as operações de sincronização junto ao repositório por meio das ferramentas clientes.

Nos testes realizados verificou-se que só o *Subversion* possui *commit* atômico, pois ao ser submetido um conjunto de modificações ao repositório por meio de um *commit*, caso um dos arquivos esteja bloqueado para edição por outro usuário, o *CVS* permite que seja aceito apenas uma parte das alterações, já o *Subversion* não permite, garantido a atomicidade do *commit*, além de seus números de revisões estarem atrelados a cada operação de *commit* e não aos arquivos.

Para analisar o controle de diferentes tipos de arquivos foram armazenados arquivos com extensões *.jude* e *.php*, contendo diagramas de caso de uso e códigos-fonte, respectivamente. Enquanto o *CVS* necessita que haja ajustes tanto no cliente quanto no servidor para realizar o armazenamento de arquivos binários, o *Subversion* proporciona o versionamento de diferentes tipos de arquivos, inclusive aos arquivos binários, sem necessidade de configurações específicas.

Quanto ao controle de mudanças ambos apresentam eficiência; os dados de identificação das ações de *commit* e *check-out* realizadas durante os experimentos ficaram registradas no servidor. Permitindo assim, que o gerente de configuração possa verificar quem, onde e o quê foi modificado. Ambos também permitem a configuração, controle de acesso (leitura e escrita) e permissão de acesso dos usuários aos arquivos no repositório.

Ao serem realizados os testes de *merges* e *braches* ambos os sistemas também apresentaram tais funcionalidades que permitem a criação e junção das versões dos arquivos. Quanto à renomeação de arquivos, o CVS, ao contrário do *Subversion*, não possibilita nenhuma forma para realizar tal funcionalidade.

A instalação e utilização de ambos foram testadas apenas nos ambientes: *Linux* e *MS Windows*. Todavia, as documentações de ambas apontam também a compatibilidade com outros ambientes operacionais tais como *Apple Mac OS* e *Sun Solaris*.

Tabela 3  
Comparativo do critério: características e funções

Características/ Funções	CVS	Subversion
Commit atômico	Não possui	Possui
Controle de diferentes tipos de arquivos	Possui	Possui
Controle de Mudanças	Possui	Possui
Controle de Permissões	Possui	Possui
Merges e Braches	Possui	Possui
Portabilidade	Possui	Possui
Renomeações de arquivos	Não possui	Possui
Suporte	Possui	Possui

Na tabela 3 é apresentado o comparativo das características e funções entre os sistemas analisados, atribuindo as indicações “Possui” ou “Não possui”.

Para o critério características e funções o *Subversion* apresentou melhor resultado, atendendo a todos os itens analisados, enquanto o CVS não apresentou: *commit* atômico e renomeações de arquivos.

O suporte é atendido, pois ambos permitem que o usuário possa identificar as versões e realizar recuperações junto ao repositório. O usuário pode utilizar, por exemplo, a funcionalidade *diff*, que permite comparar a cópia do arquivo presente na *workspace* com a cópia que está no repositório ao realizar um *merge*. A fig. 1 mostra a tela de *diff* do arquivo-fonte *index.php*, onde no lado esquerdo (*Working Base, Revision 2*) é apresentada a última versão do arquivo no repositório, e do lado direito (*Working Copy*) é apresentado o arquivo destacando a alteração realizada pelo usuário.

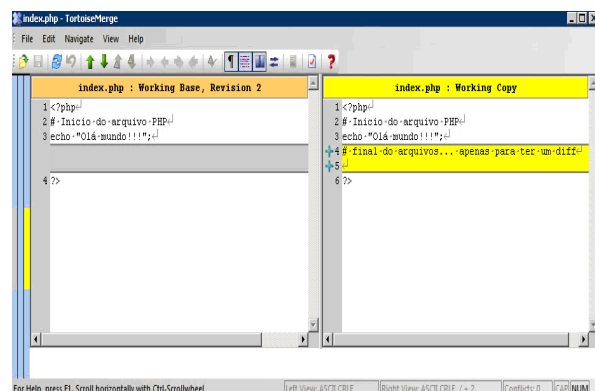


Fig. 1. Tela de *diff* do TortoiseSVN

Ambos os sistemas não apresentaram problemas na integração com as ferramentas clientes *TortoiseCVS* e *TortoiseSVN*, as quais auxiliam na utilização dos sistemas, possibilitando uma melhor aplicabilidade no processo de desenvolvimento dos *softwares*.

Os dois sistemas apresentam características próprias, e a extensibilidade não é aplicada pelos mesmos, embora ambos pertençam a projetos de *software* livre, sendo possível o usuário oferecer sua contribuição de melhoria às equipes de desenvolvimento dos sistemas.

Os testes realizados das características e funções utilizaram o conceito cliente/servidor, comprovando assim que ambos os sistemas podem ser empregados nesse tipo de ambiente e dessa forma garantir o suporte para uma equipe geograficamente distribuída.

Ambos os sistemas utilizam linha de comando, e apresentam melhor usabilidade ao serem integradas com ferramentas clientes, apresentando assim uma melhor *interface* para o usuário.

Ambos os sistemas analisados e suas respectivas versões (CVS 1.11.23 e SVN 1.6.5), apresentam um histórico de correções realizadas ao longo do desenvolvimento dos sistemas, o que se pressupõe a existência de uma maturidade para sua utilização em projetos de *softwares*.

Quanto ao suporte a configurações, ambos disponibilizam documentações nos sítios dos seus grupos mantenedores, que possibilitam a configuração e utilização de suas funções.

Por ambas se tratarem de sistemas baseados em código aberto, não possuem treinamento oferecido pelo fornecedor na aquisição do produto. Entretanto, várias empresas oferecem treinamento para a utilização desses sistemas, e na *web* é possível encontrar tutoriais que auxiliem o usuário na configuração e uso dos sistemas e ferramentas clientes.

Na tabela 4 é apresentado o comparativo final, com todos os critérios estabelecidos para a análise, onde cada sistema é classificado como: “Não atende”; “Atende parcialmente” e “Atende Totalmente”.

O comparativo realizado demonstra que o sistema *Subversion* atendeu a um maior número dos critérios adotados. Embora o *Subversion* atenda parcialmente o critério de usabilidade, e não atenda aos critérios de extensibilidade e treinamento, atende totalmente a todos os outros critérios estabelecidos, enquanto o sistema CVS, além de não atender aos critérios de extensibilidade e treinamento, atende parcialmente ao critério de usabilidade e ao critério de características e funções, o qual se apresenta como essencial para um bom desempenho do sistema em projetos de *softwares*.

Tabela 4  
Comparativo dos critérios estabelecidos

Critérios	CVS	Subversion
Características e funções	Atende parcialmente	Atende totalmente

Integração	Atende totalmente	Atende totalmente
Aplicabilidade	Atende totalmente	Atende totalmente
Extensibilidade	Não atende	Não atende
Suporte a equipe	Atende totalmente	Atende totalmente
Usabilidade	Atende parcialmente	Atende parcialmente
Maturidade	Atende totalmente	Atende totalmente
Suporte a configurações	Atende totalmente	Atende totalmente
Treinamento	Não atende	Não atende

## 6. Considerações

A utilização de sistemas de controle de versões é uma atividade essencial da gerência de configurações, para realizar monitoramento das mudanças realizadas em projetos de desenvolvimento de *softwares*, e controlar o desenvolvimento paralelo. Dentre as soluções existentes atualmente para realizar o controle de versões, merecem destaque o *CVS* e o *Subversion*, objetos desta pesquisa, por ambos serem baseados em código aberto, possuírem qualidade e notoriedade no uso em projetos de *softwares*. Nesta pesquisa, foi realizada uma análise comparativa baseada nos preceitos definidos pelo *RUP* para a escolha de ferramentas de suporte utilizadas em desenvolvimento de *softwares*, com o intuito de definir quais características cada sistema apresentaria de forma a contribuir para a qualidade em controle de versões.

Ao ser realizado o comparativo entre as duas ferramentas, foi observado que o *Subversion*, em relação ao *CVS*, atende um maior número dos critérios adotados pela análise, destacando-se quando se trata de atomicidade de *commits*, renomeações de arquivos e controle de diferentes tipos de arquivos. Podendo o mesmo, ser uma boa indicação para a gerência de configuração no momento da escolha de um sistema de controle de versões. Sabendo que ambos os sistemas encontram-se com o desenvolvimento contínuo, é possível que as características e funções não apresentadas no *CVS*, possam vir a serem implementadas pelas comunidades mantenedoras dos sistemas.

Embora os itens adotados para a análise de características e funções tenham sido definidos a partir de comparativos de pesquisas já realizadas, os tipos dos testes realizados não permitem generalizações quanto ao resultado. As percepções obtidas neste trabalho sobre a importância do controle de versões em projetos de *softwares* podem ser complementadas com trabalhos futuros que possibilitem a análise dos impactos ocorridos em um ambiente de desenvolvimento após a adoção do *Subversion* como sistema de controle de versões em um projeto de *software* legítimo, já que a presente pesquisa apenas utilizou-se de testes unitários em arquivos genéricos, com a pretensão de analisar as principais características dos sistemas.

## 7. Referências

[1] Ataídes, A. C. (2006). Um método para acompanhamento e controle da implantação do CMMI. 127f. Dissertação (Mestrado em Engenharia Elétrica) – ENE/FT/UnB, Brasília.

[2] Berlack, H. R. (1991). Software Configuration Management. 2ed. Estados Unidos: Wiley.

[3] Bolinger, D., Brosson, T. A. (1995). RCS and SCCS - From Source Control to Project Control. <http://www.oreilly.com/catalog/rscs/chapter/index.html> (acessado em 30-05-2009).

[4] Bon, J. V. (2005). Foundations of IT Service Management, based on ITIL. Holanda: Van Haren Publishing.

[5] Carneiro, A. N. N. (2007). Um guia para controle de versão de Projeto de Software. 43f. Monografia (Graduação em Engenharia da Computação) – Escola Politécnica de Pernambuco, Recife.

[6] Cia, T. M. (2006). Modelo de avaliação do processo de gerência de configuração de software. 116f. Dissertação (Mestrado em Ciências da Computação e Matemática Computacional) – ICMC/USP, São Carlos.

[7] Collins-Sussman, B., Fitzpatrick, B. W., Pilato, C. M. (2007). Controle de versão com subversion. EUA: O'Reilly, 2007. <http://www.svnbook.read-bean.com/svnbook/book.html> (acessado em 06-09-2009).

[8] Tavares, R. P. (2007). Um processo de gerenciamento de configuração de software com repositório de artefatos definidos dinamicamente. 174f. Dissertação (Mestrado em Computação Aplicada) – INPE, São José dos Campos.

[9] Glasser, A. L. The evolution of a source code control system (1978). Proceedings of the software quality assurance workshop on Functional and performance issues, [S.I.s.n.], p. 122-125.

[10] Junqueira, D. C. (2007). Um controle de versões refinado e flexível. 2007. 86f. Dissertação (Mestrado em Ciências da Computação e Matemática Computacional) – ICMC/USP, São Carlos.

[11] Midha, A. K. (1997). Software configuration management for the 21st century. Bell Labs Technical Journal. [http://citeseer.nj.com/midha97\\_software.html](http://citeseer.nj.com/midha97_software.html) (acessado em 30-05-2009).

[12] Moro, R. G. (2003). Uma infraestrutura para controle de versões e adaptações de páginas Web. 2003. 78f. Dissertação (Mestrado em Ciências da Computação) – PPGC/UFRGS, Porto Alegre.

[13] Murta, L. G. P. (2006). Gerência de configuração no desenvolvimento baseado em componentes. 213f. Tese (Doutorado em Engenharia de Sistemas e Computação) – COPPE/UFRJ, Rio de Janeiro.

[14] Oliveira, S. B. C., Tanaka, A. K., Vianna, D. S. (2006). Avaliação de ferramentas para controle automatizado de versões de artefatos de requisitos de software. WER06-Workshop em Engenharia de Requisitos, Rio de Janeiro. [http://wer.inf.puc-rio.br/WERpapers/artigos/artigos\\_WER06/susana\\_oliveira.pdf](http://wer.inf.puc-rio.br/WERpapers/artigos/artigos_WER06/susana_oliveira.pdf) (acessado em 30-05-2009).

[15] Oliveira, H. L. R. O. (2005). Odyssey-VCS: Uma abordagem de controle de versões para elementos de UML. 94f. Tese (Mestrado em Engenharia de Sistemas e Computação) COPPE/UFRJ, Rio de Janeiro.

[16] Pressman, R. S. (2006). Engenharia de Software. 5ª. ed., Rio de Janeiro: Mc Graw-Hill.

[17] Rentes, M. (2009). Subversion: Controle total sobre o software – 2ª Parte. Revista Programar, Portugal, a.3, n.18, p.14-21.

[18] Sampieri, R. H., Collado, C. F., Lucio, P. B. (2006). Metodologia de pesquisa, 3ed. São Paulo: McGraw-Hill.

[19] Sant'anna, N. (2000). Um ambiente integrado para o apoio ao desenvolvimento e gestão de projetos de software para sistemas de controle de satélite. São José dos Campos: INPE. <http://urlib.net/dpi.inpe.br/lise/2002/03.28.20.03> (acessado em 30-05-2009).

[20] Santos, R. B. S. (2007). Sistema de controle de versões para edição cooperativa de vídeo MPEG-2. 110f. Dissertação (Mestrado em Informática) - PUC, Rio de Janeiro.

[21] Silva, F. A. (2005). Um modelo de dados temporal orientado a objetos para gerenciar configurações de software. 99f. Dissertação (Mestrado em Ciências da Computação) - UFRGS, Porto Alegre.

[22] Silva, S. R. Q. M. (2005). Controle de versões – um apoio à edição colaborativa. 88f. Dissertação (Mestrado em Ciências da Computação) – ICMC/USP, São Carlos.

[23] Soares, M. D. (2000). Gerenciamento de versões de páginas web. 102f. Dissertação (Mestrado em Ciências da Computação e Matemática Computacional) – ICMC/USP, São Carlos.

[24] Sommerville, I. (1995). Software Engineering, 5ed. Addison-Wesley.